Práctica 5:

ÁRBOLES BINARIOS DE BÚSQUEDA

OBJETIVO

- Implementación de árboles binarios de búsqueda en C++.
- Uso de los mapas de la STL.

MATERIAL A ENTREGAR

- abb.h, abb.cpp, test_arbol.cpp, test_map.cpp
- makefile
- Ejercicios apartado 1. **(Previo)**

DESARROLLO

1. Ejercicios previos

Escribe el código en C++ del método tam() de la clase ABB (definida en abb.h).

2. Clase ABB

Implementar la clase ABB a partir de abb.h, las transparencias de clase y el capítulo 12 del Nyhoff. Si se realiza algún cambio en abb.h se deberá justificar adecuadamente. La clave deberá ser un string, el dato a guardar será un objeto de tipo *Alumno*. La definición, junto con algunas funciones útiles se encuentra en *alumno.h* y *alumno.cpp*.

3. Prueba de la clase Arbol

Para comprobar el buen funcionamiento de la clase ABB, se deberá escribir un programa (**test_arbol.cpp**) que genere un ABB con 1.000 alumnos. Después se deberán realizar los siguientes pasos:

1. Mostrar el árbol en pantalla
2. Buscar 3 DNI. 1 de ellos no debe existir. Si el DNI existe, mostrar el alumno por pantalla.
3. Mostrar el **tamaño** y la **altura**.
4. Mostrar la **media de los DNI** y **cuantos DNI comienzan por 30** en el árbol generado.

La función que calcula los DNI que comienzan con 30 no deberá buscar más ramas de las necesarias dentro del árbol. Para el cálculo de la media de los DNI se deberá eliminar el último carácter del DNI y convertirlo entonces a número para poder calcular la media.

¿El árbol generado está equilibrado? Indicar cual debería ser la altura aproximada para un árbol equilibrado.

4. La clase *map* de la STL

La STL ofrece una clase muy similar a la clase ABB que has escrito. Es la clase map.

Rescribe el programa del apartado 3 (**test_map.cpp**) usando la clase map en lugar de ABB y comprueba que funciona correctamente. No se deberá mostrar la altura y en lugar de decir cuántos DNI comienzan por 30, se deberán mostrar por pantalla (con el menor coste posible).